Execution Models: A Bottom-Up Approach (EMBU)

ASCR/ASC Exascale PI Meeting
Portland, OR

April 17, 2012

Sandia National Laboratories

Robert Clay

Mike Heroux

Gilbert Hendry

Joe Kenny

Lawrence Berkeley National Laboratory

David Donofrio

John Shalf

Nick Wright

Indiana University

Thomas Sterling

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Outline

- Project overview
- Key abstractions
- Our approach
- Initial results
- Summary

Sandia National Laboratories

Execution model bottom-up study overview

Examine potential execution models and impact on exascale

Bottom-up approach: start with concrete examples of execution models and hardware

Split into two phases (synchronized with top-down):

- Phase 1: rapid co-design iteration to develop a whitepaper in the Feb 2012 timeframe
- Phase 2: slower, deeper iteration incorporating additional execution models and applications over following 2.5 years
- However: output is continuous
  - collaboration tools used to continuously update living documents
  - progress closely shared with DOE and the top-down project

Develop an Execution Model Toolkit (EXEMT)

- Collection of coarse- and fine-grained components for studying execution models

Demonstrate and document a methodology that can be applied to additional execution models

Execution Model Definition

An execution model is a paradigm of computing establishing the principles of computation that govern the interrelationships of the abstract and physical components and their functions comprising the computational process.

Execution models differ by the way they project the abstract computation on to the physical computing medium guiding:

- **The programming model semantics,**
- **The physical machinery structures and mechanisms, and**
- **The policies and methodologies resource management and task scheduling embodied in the system software (runtime and operating systems).**

An execution model is a conceptual tool for the co-design and interoperability of the system layers exposing the "decision chain" that establishes the responsibilities of each layer in contributing to the determination of which actions are performed on what objects, where, and when.

Key questions EMBU is addressing

At what point do you decide to move the work to the data (or the reverse)

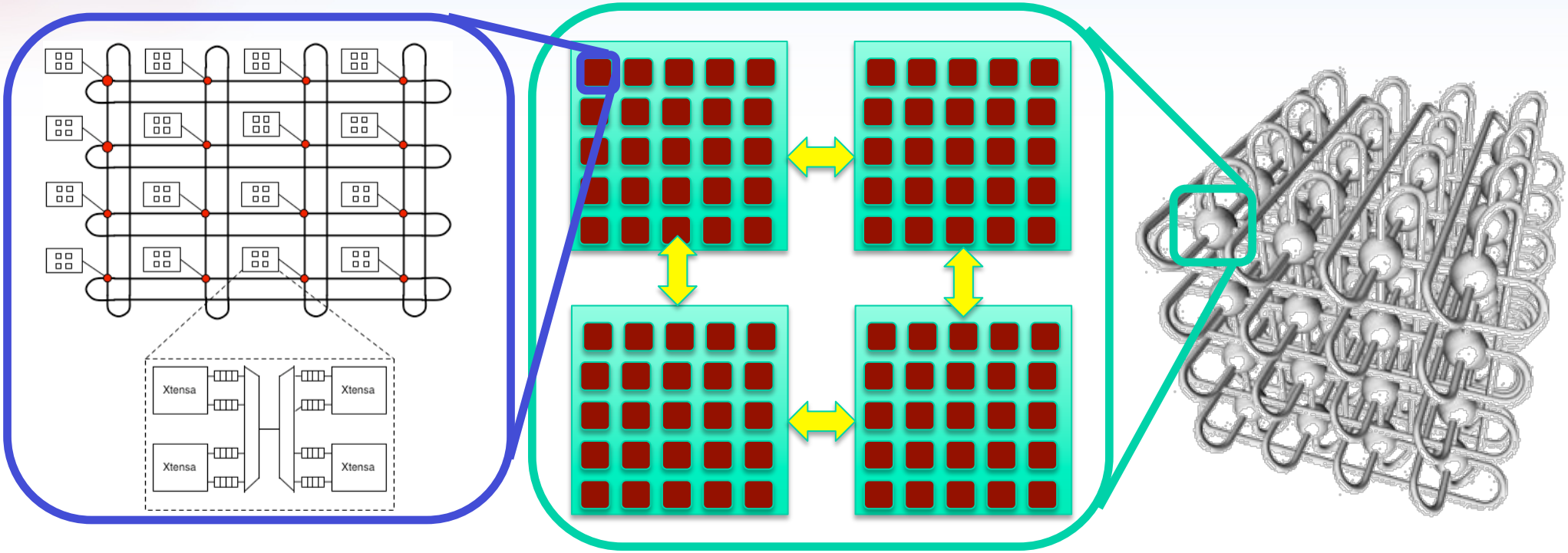
- who makes this decision - exec model or programmer?
- Does PX have sufficient info to make this decision?

Can PDE solves and block structured grids be efficiently scheduled as a dataflow rather than a SPMD?

Overall, will a new execution model make it easier to map a problem onto future machines

- Will mapping be easier to reason about?
- Will mapping be more performance portable?
- Will overheads of implementation or hardware requirements undercut the benefits of the new approach?

Notional multi-scale machine abstract model



- Cores (many simple cores)
  - Flat clock rate
  - Multithreaded (n-threads)
  - SIMD (n-slots)
  - Fat+Thin cores (ratio)
- NoC
  - Constrained Topology (2D)
- Cache Hierarchy (size, type, assoc)
  - Automatic caches
  - Scratchpad/software managed
  - NVRAM
  - Alternative coherency methods

- Non-uniform memory access (NUMA) between cores and memory channels
  - Topology may be important
  - Or perhaps just distance
- Memory
  - Increased NUMA domains
  - Intelligence in memory (or not)
- Fault Model for node
  - FIT rates, kinds of faults, granularity of faults/recovery

- Interconnect
  - Constrained Topology (Torus, Tapered Dragonfly)
  - Bandwidth/latency/overhead for communication
- Primitives for data movement/sync
  - Global Address Space or messages only
  - Memory fences
  - Transactions / remote atomics

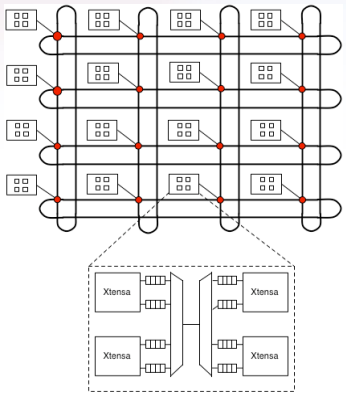
Node-level models

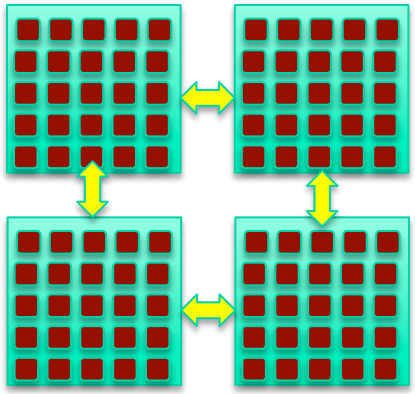
- Node-level execution model simulation: Develop a node-level implementation of an execution model capable of running the POP and GTC surrogates.
  - Implement node-level EXEMT: Develop a node-level execution model toolkit (EXEMT). It will be rich enough to support execution model co-design exploration, yet simple enough to be implemented in the short time frame allocated to this task. The work will be performed using the ACE simulation environment.
  - Implement mini-app node codes: Implement node-level surrogates for POP and GTC using the EXEMT suitable for running in the ACE simulation environment.
  - *[Milestone]* Demonstrate node-level mini-app simulations: Demonstrate that the EXEMT-based application surrogates run in the ACE simulation environment.

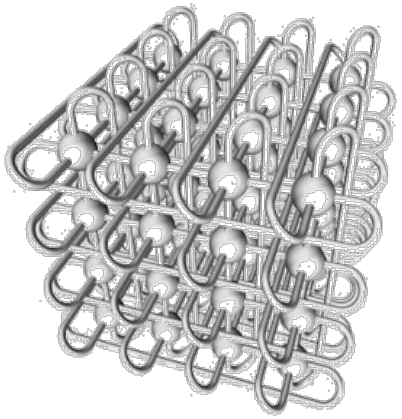
Network-level models

- Network-level execution model simulation: Develop a network-level execution model toolkit (EXEMT). It will be rich enough to support execution model co-design exploration, yet simple enough to be implemented in the short time frame allocated to this task. The work will be performed using the SST/macro simulation environment.
  - Implement network-level EXEMT: Develop a feature set for off-node aspects of the execution model. These will be implemented as abstracted models in SST/macro and must be sufficiently complete to support the mini-applications used in the EMBU project.
  - Implement mini-app skeleton codes: Implement the EXEMT-based mini-application skeleton codes, which can drive the SST/macro simulators. Models for the on-node portion of computations will be derived from the node-level effort.
  - *[Milestone]* Demonstrate network-level mini-app simulations: Demonstrate that the EXEMT-based mini-application skeletons run in the SST/macro simulator.

Mapping of simulation tools into machine abstract architecture







RAMP/GreenFlash: Chip Level Simulation

- Extend GreenFlash/RAMP simulation for more general proxy model (lego blocks for rapidly prototyping chip models)
- Create parameterized NoC and memory hierarchy
- Provides model-checking for energy models offered by software simulators (it is a *real* circuit design… not a model thereof)

SST: simulation of different interconnect architectures

- Driven by input traces or skeletonized code (either manually or via ROSE)
- Use reduced node model to bridge gap between full cycle-accurate model for the chip

Parameterizations

Network Workload

Sandia National Laboratories

Modeling & Simulation as a Co-design Tool

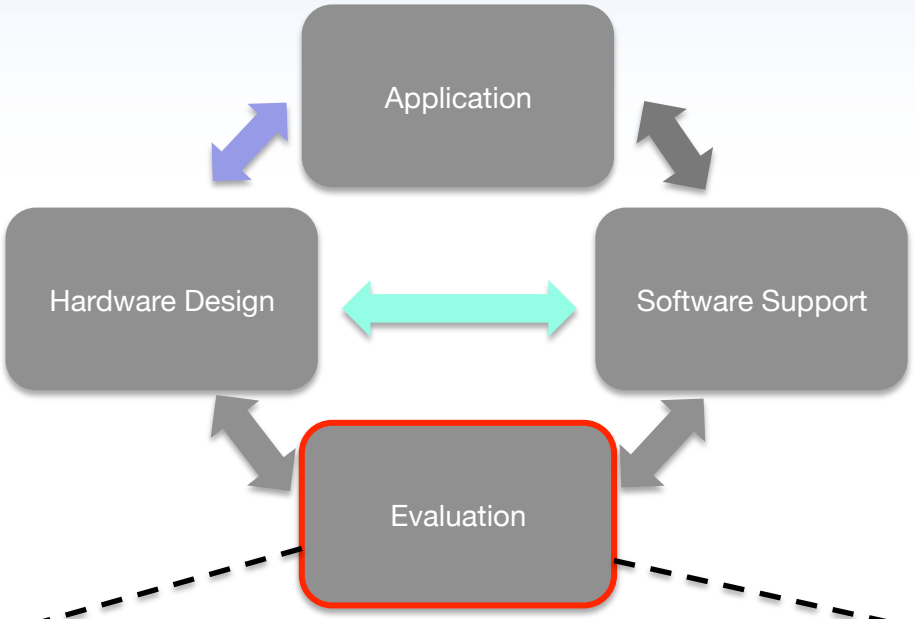
Ultimate Question:

Do my applications run well on the machine?

Intermediate Questions:

Is the application programmed in the best way?

Is there a good mapping of hardware support for software



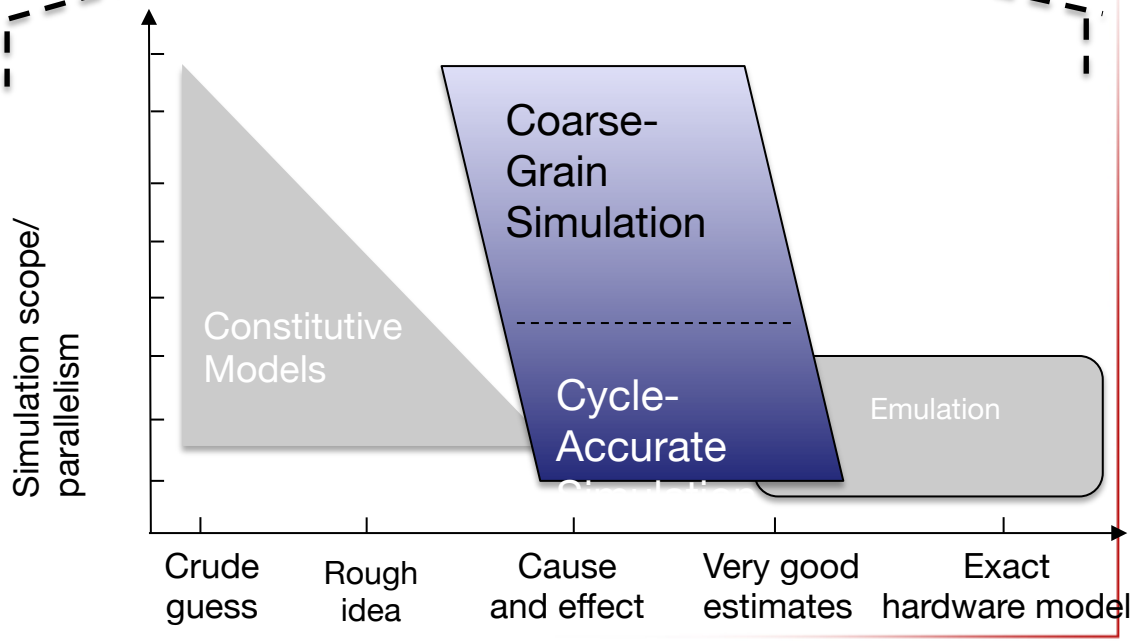
———— Evaluation ————

Constitutive Models – can be powerful, but hard to investigate new concepts and complex interactions

Coarse-Grained Simulation – accurate, predicts trends, can scale

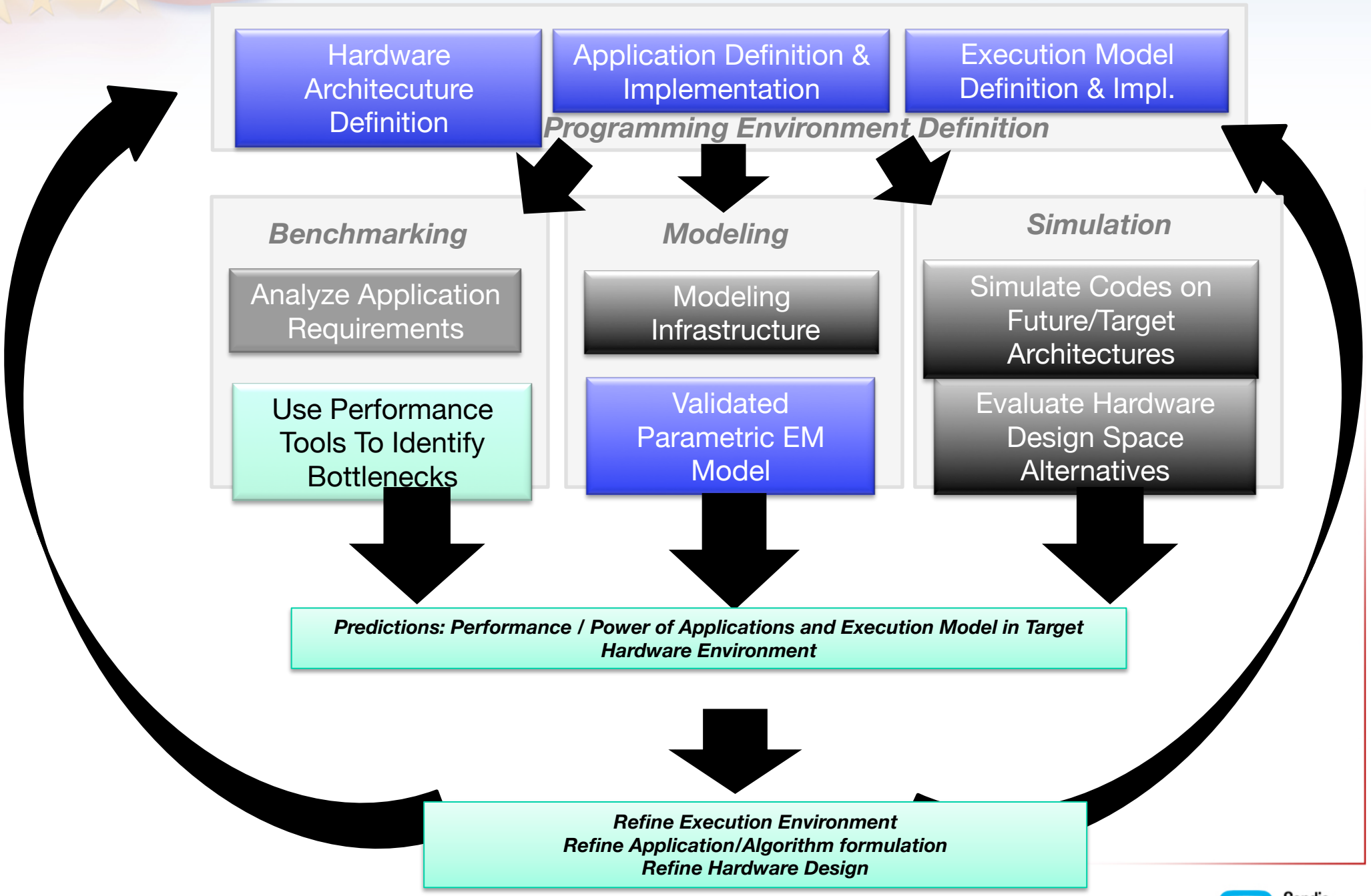
Cycle-Accurate Simulation – highly accurate, but can only scale so far

Emulation – essentially exact and fast, but expensive



Sandia National Laboratories

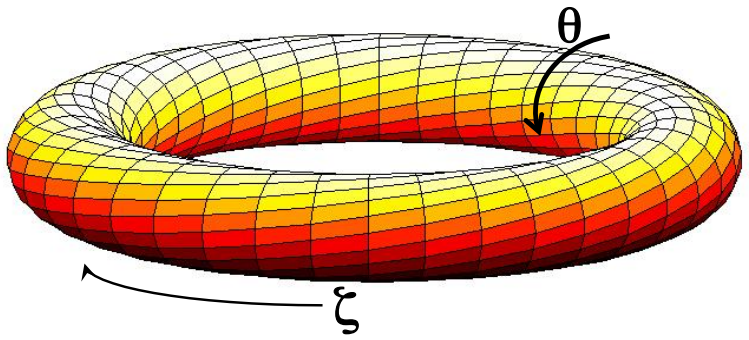
Execution Models in the Design Loop



Sandia National Laboratories

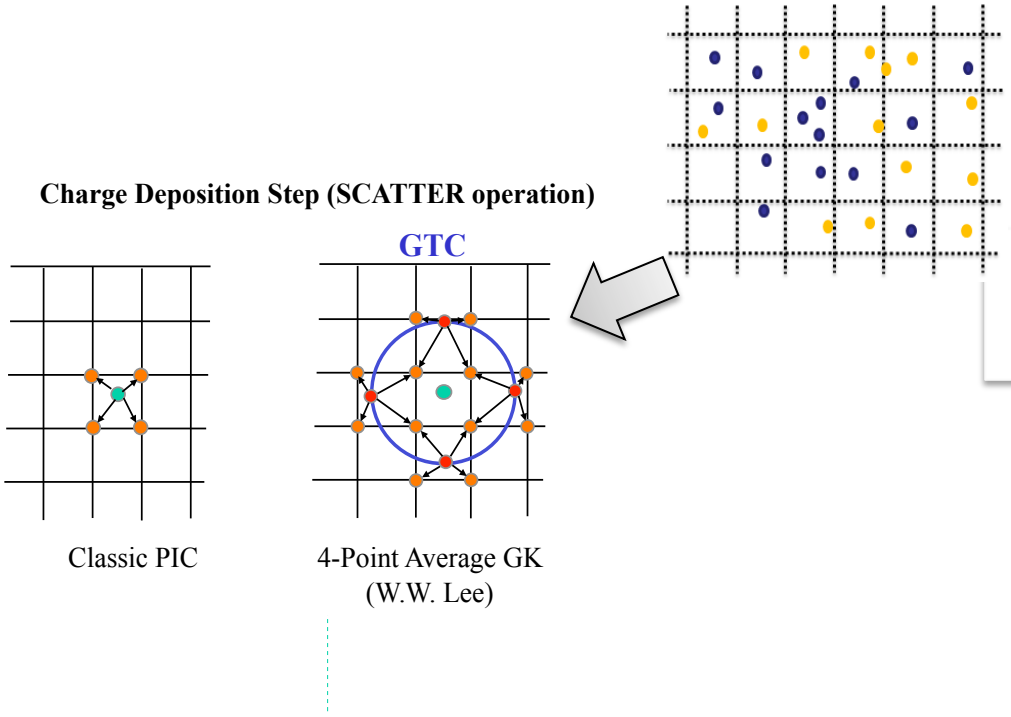
Starting with the Gyrokinetic Toroidal Code

- GTC uses PIC method to simulate plasma microturbulence for fusion devices
- Written in F90 with MPI
- rs



- Grid memory accesses depend on the order in which particles are processed.
- In a multithreaded implementation with a shared grid, multiple threads update grid locations in parallel.

similar to the GUPS benchmark. However, implementations usually exploit the fact that PIC is a physical many-body simulation method.

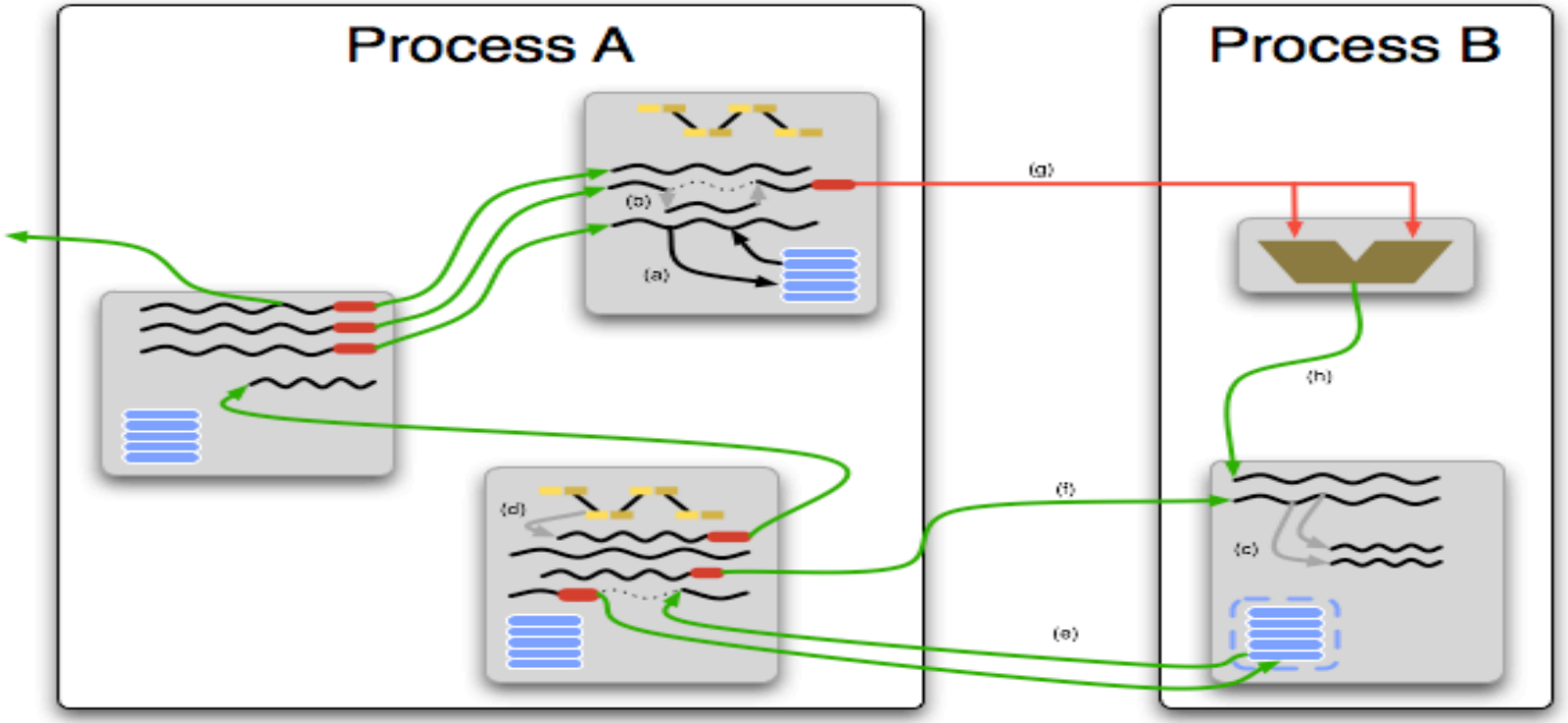


Sandia National Laboratories

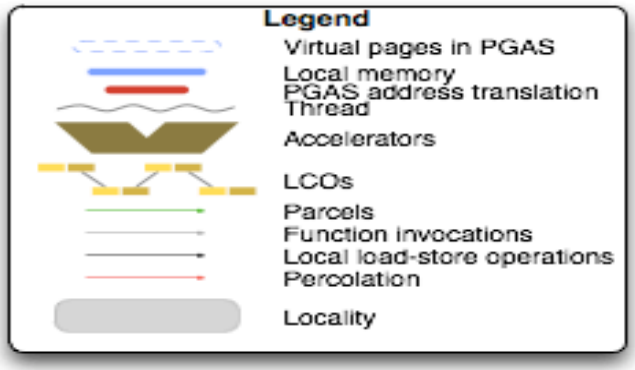
A ParalleX Review

1. Synchronous Domains
2. AGAS – Active Global Address Space
3. ParalleX Processes – with capabilities protection
4. Computational Complexes – threads & fine grain dataflow
5. Local Control Objects – synchronization and global distributed control state
6. Distributed control operation – global mutable data structures
7. Parcels – message-driven execution and continuation migration
8. Percolation – heterogeneous control
9. Micro-checkpointing – compute-validate-commit
10. Self-aware – introspection and declarative management

ParalleX Model



(a) Local data access
(b) Local thread invocation (co-routine)
(c) Local thread invocation (concurrent threads)
(d) LCO spawning a thread
(e) Remote atomic memory operation through parcels
(f) Remote thread invocation through parcels
(g) Percolation
(h) Thread creation as result of continuation action

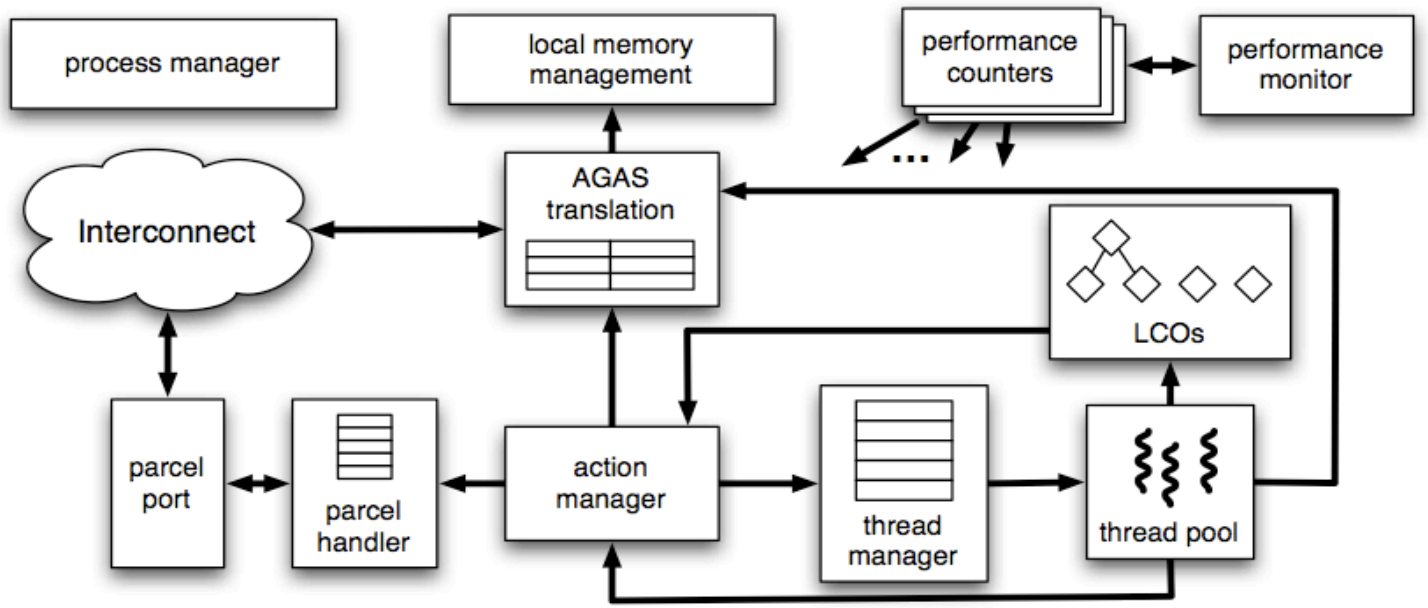


Sandia National Laboratories

HPX Runtime Design

Current version of HPX provides the following infrastructure as defined by the ParalleX execution model

- **Complexes (ParalleX Threads) and ParalleX Thread Management**
- **Parcel Transport and Parcel Management**
- **Local Control Objects (LCOs)**
- **Active Global Address Space (AGAS)**



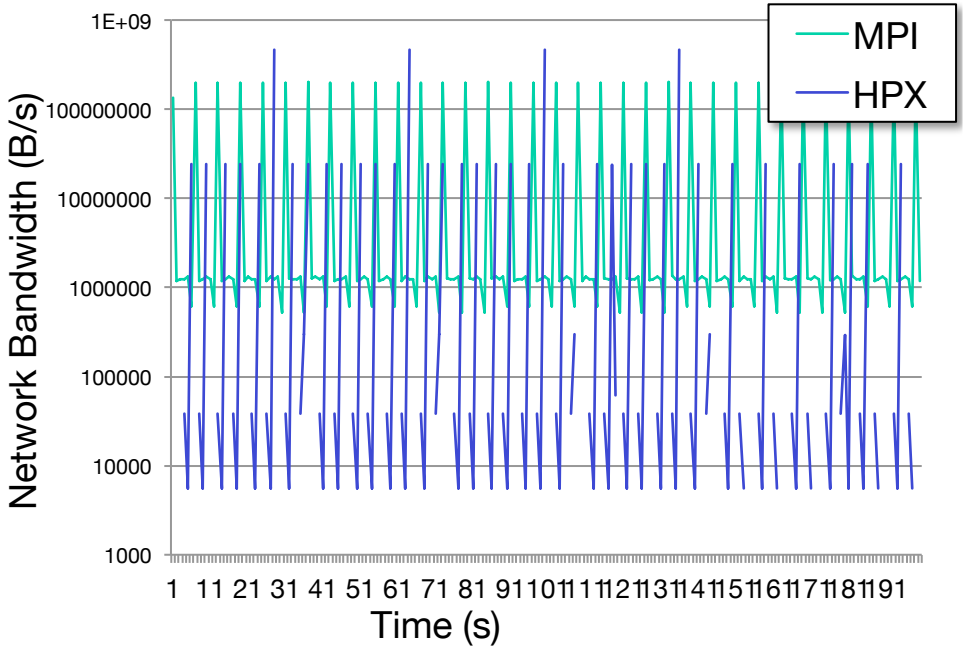
Sandia National Laboratories

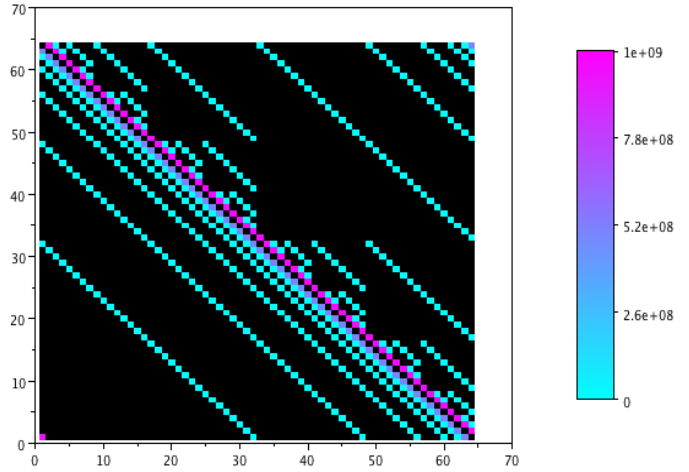
GTC with static MPI vs. dynamically scheduled HPX

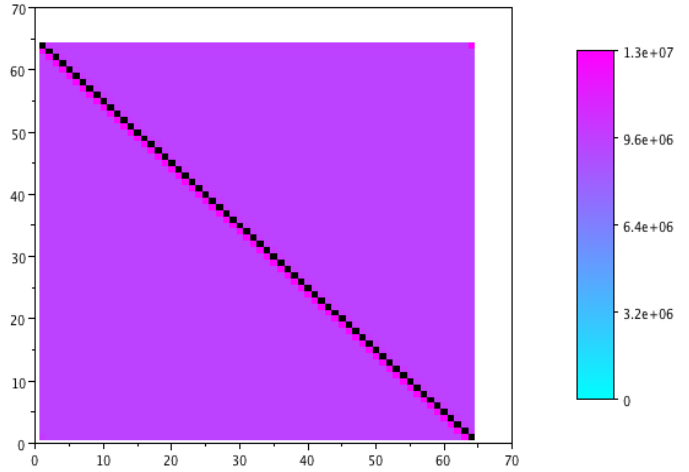
Preliminary experiments show asynchronous scheduling (HPX) changes the communication pattern vs. MPI

.

Asynchronous communication (HPX) uses many more, much smaller messages, but less aggregate network bandwidth.





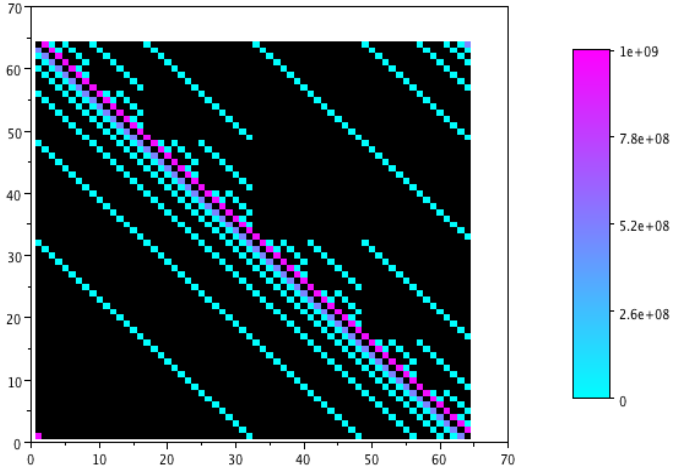


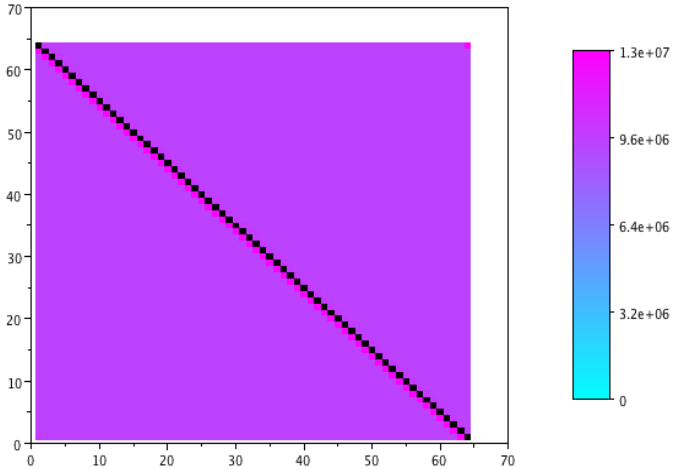
Sandia National Laboratories

Accelerating HPX

- HPX leverages a massive threading model to hide latency
  - Threads can be dynamically created and transmitted across localities
  - Hard limit of one thread per core
- High frequency and widely distributed communication (compared to MPI)
  - Central to HPX goal of moving the work to the data rather than the reverse
  - Communication consists mostly of small packets
    - Keeps total bandwidth requirements reasonable

What hardware constructs can accelerate the thread creation and transfer in HPX?





Sandia National Laboratories

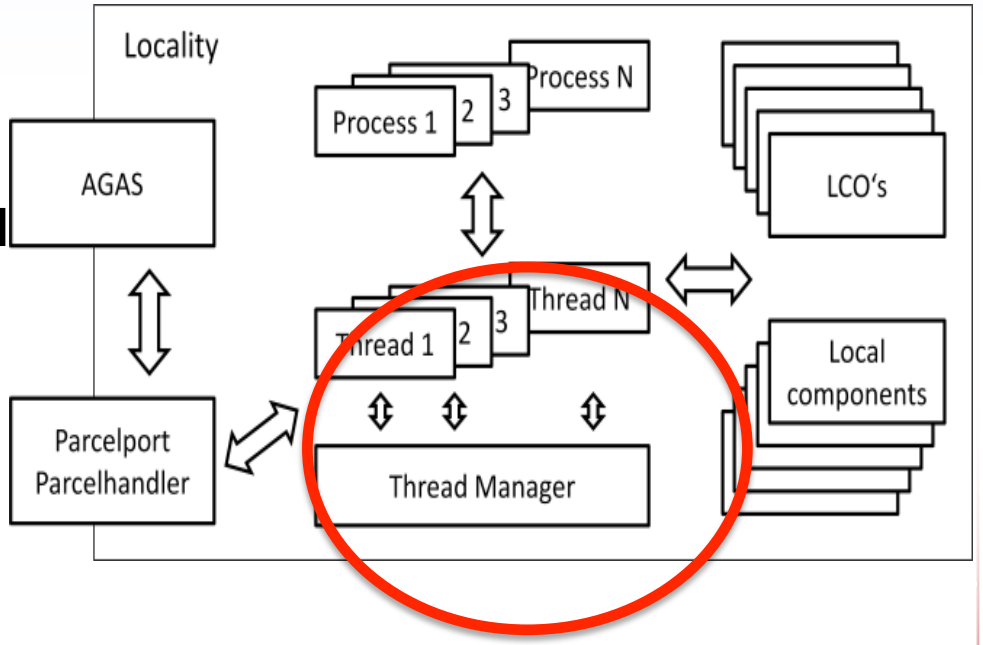
Accelerating HPX – Thread Management

Option 1: Double Buffering

- Load a future thread's context (in the background) while the active thread is executing
- Software controlled memory attached to processor can hold local thread context

Option 2: Hardware Threads

- Build cores with multiple hardware threads that dynamically context switch depending on resources
- HPX "one thread per-core" model preserved
- Allows greater latency hiding as more threads will be ready to execute



- **Co-Design Opportunities:**
  - Size of local store required for complete thread context?
  - DMA engine attached to memory can support rich thread transfer commands to reduce burden on processor

Sandia National Laboratories

Summary

- We are simulating the performance of 'complete' systems, and are beginning to collect the data needed to make design decisions for Exascale systems. This is co-design.

- These results will allow us to quantify design tradeoffs associated with technical challenges such as starvation, latency, overhead, and delays due to contention as well as the practical constraints of power, reliability, generality, and programmability.

- We are building tools and methodology to assess new paradigms in the form of new execution models to exploit runtime information, manage asynchrony, co-design processor architectures and applications, expose untapped logical and physical parallelism, and ensure continued operation by graceful degradation.